Coattention, Dynamic Pointing Decoders & QANet for Question Answering

Stanford CS224N Default Project: : IID SQuAD track

Zhen Li
zhnli@stanford.edu

Mehrad Khalesi
mkhalesi@stanford.edu

Jeffrey Sun
jeffsun1@stanford.edu

Abstract

The task of question answering (QA) requires language comprehension and modeling complex interaction between the context and the query [1]. Recurrent models primarily use recurrent neural networks (RNNs) to process sequential inputs, and attention component to cope with long term interactions [2]. However, recurrent QA models have two main weaknesses. First, due to the single-pass nature of the decoder step, models have issues recovering from incorrect local maxima. Second, due to the sequential nature of RNNs these models are often too slow for both training and inference. To address the first problem, we implemented a model based on Dynamic Coatention Network (DCN) that incorporates a dynamic decoder that iteratively predicts the answer span [3]. To improve the model efficiency, we also implemented a transformer based recurrency-free model (QANet), which consists of a stack of encoder blocks including self-attention and convolutional layers. On the Stanford Question Answering Dataset (SQuAD 2.0), our best QANet based model achieved 68.76 F1 score and 65.081 Exact Match (EM) on the dev set and 66.00 F1 and 62.67 EM on the test set.

1 Introduction

There has been growing research interest in machine comprehension and question answering tasks, which are challenging problems with wide real world applications like search engine retrieval. The closed-domain question answering task involves giving the model two sequences, the context and the query, and the model predicts the start and end positions of the correct answer's span in the context. The SQuAD [4], which contains 100K+ questions, has become a popular data set for this task. Recently great progress has been made on question answering by using neural attention mechanisms [5]. This approach enables the system to focus on a targeted area within a context paragraph that is most relevant to the answer of the query.

In the early works, the attention weights are uni-directional extractions from fixed-size context vectors, and they are dependent on the attention values from previous time steps. A bi-directional attention flow was proposed in the BiDAF model [1], which does not use fixed-size context vectors and instead computes attention weights at each time step, only relying on the context and query at the current time step. BiDAF out-performs previous models by avoiding information loss caused by early summary and forces the attention and modeling layers to learn the interaction between the context and query.

Later on, Dynamic Coattention Network (DCN) [3] proposed a coattention encoder that learns the co-dependent context and query representations, and a dynamic decoder which alternates between estimating the start and end of the answer span. DCN achieved state-of-the-art results on SQuAD.

Both BiDAF and DCN consist of a recurrent model (RNN) to process sequential inputs. RNN based models are usually slow for both training and inference due to their recurrent nature. To improve model efficiency and adopt parallelism QANet [2] proposed a recurrency-free model. QANet uses encoder blocks that consist of convolution and self-attention to encode the context and query. Interactions between context and query are learned using standard attention. The idea of QANet is to use convolution to capture the local structure of the text while self attention learns the global interaction between each pair of words, and the feed-forward based architecture can improve model efficiency significantly.

In this project, we implemented dynamic coattention mechanisms from DCN and the transformer-based model from QANet. We trained both models on SQuAD data set and ran evaluations to obtain the EM and F1 stores. We compared DCN with BiDAF baseline model. We also demonstrated the improvement of QANet over recurrent models in terms of

speed and accuracy. We experimented with the combination of various techniques including character level embedding and enhanced input features, while training the model with different hyper-parameters and performing model analysis.

2 Related Work

Before the use of neural attention models, statistical approaches were developed to address question answering. One proposal came from Wang et al. (2015) where semantic and syntactic features were used in a statistical model [6]. Along similar lines, Chen et al. (2016) produced a competitive statistical model based on lexical, syntactic, and word order features [7]. Neural attention has since gained greater research relevance, with Kadlec et al. (2016) presenting a single attention step pointer mechanism [8]. Sordini et al. (2016) extended attention to be iterative [9]. Notably, Lu et al. (2016) achieved state-of-the-art results in visual question answering by using a hierarchical coattention model [10]. For the SQuAD dataset, Wang & Jiang (2016) showed that a Match-LSTM encoder could be used [11]. Furthermore, Vinyals et al. (2015) introduced a pointer network decoder [12]. The general research points to attention being a strong candidate for question answering, and coattention for having mapping characteristics that allow for affinities of the document and question to be taken into account. Furthermore, there appears to be ongoing research in improving the pointer decoder architecture which selects the answer.

3 Approach

3.1 Baseline

Our baseline model is based on Bidirectional Attention Flow (BiDAF) [1]. The source code is from GitHub repository: `https://github.com/minggg/squad.git`.

3.2 DCN Overview

The DCN model exhibits a modular architecture consisting of 4 main layers: embedding layer, document and question encoder, contention encoder and dynamic pointing decoder as presented in the dynamic coattention network (DCN) model [3]. The contention encoder captures the interaction between the question and the context and a dynamic pointing decoder that iteratively estimates the answer span. We briefly describe our achievements in implementing these three modules, along with their functions.

3.2.1 Embedding Layer (character embeddings)

As described in Seo et al 2016 [1], we adopt the standard technique to obtain the embedding of each word w by concatenating its word embedding and character embedding. We used the pretrained GloVe vectors for word embedding with dimension of 300 [13]. We implemented character level embedding using a 1D convolution with kernel size 3 and character embedding dimension of 100, with a maxpool layer. Experiments were also performed with a kernel size of 5. We experimented with word embeddings only, character embeddings only, and word plus character embeddings. The concatenation of the character and word embedding is fed into to a two layer Highway Network [14]. Let $\boldsymbol{W}^C = (\boldsymbol{w}_1^C, \boldsymbol{w}_2^C, \ldots, \boldsymbol{w}_m^C) \in \mathbb{R}^{d \times m}$, $\boldsymbol{W}^Q = (\boldsymbol{w}_1^Q, \boldsymbol{w}_2^Q, \ldots, \boldsymbol{w}_n^Q) \in \mathbb{R}^{d \times n}$ denote the final embedding matrix for the context and question words. For our experiment we used $d = 128$ as the dimension of the embedding vector.

3.2.2 Document and Question Encoder

In this layer, we generate encoding representation of questions and contexts. Using an LSTM, the paper defines context encoding matrix $\boldsymbol{C}$ and question encoding matrix $\boldsymbol{Q}$.

$$\boldsymbol{C} = [\boldsymbol{c}_1 \cdots \boldsymbol{c}_m \, \boldsymbol{c}_\phi] \in \mathbb{R}^{l \times (m+1)} \qquad \boldsymbol{c}_i = \text{LSTM}_{enc}(\boldsymbol{c}_{i-1}, \boldsymbol{w}_i^C) \in \mathbb{R}^l$$
$$\boldsymbol{Q'} = [\boldsymbol{q}'_1 \cdots \boldsymbol{q}'_n \, \boldsymbol{q}'_\phi] \in \mathbb{R}^{l \times (n+1)} \qquad \boldsymbol{q}'_i = \text{LSTM}_{enc}(\boldsymbol{q}'_{i-1}, \boldsymbol{w}_i^Q) \in \mathbb{R}^l$$
$$\boldsymbol{Q} = \tanh(\boldsymbol{W}^{(Q)}\boldsymbol{Q'} + \boldsymbol{b}^{(Q)}) \in \mathbb{R}^{l \times (n+1)}$$

Note a non-linear projection layer on top the question encoding is introduced to allow variation between the question encoding space and the context encoding space. Also, learnable sentinel vectors $\boldsymbol{c}_\phi$ and $\boldsymbol{q}_\phi \in \mathbb{R}^l$, which allow model to not attend to any words are added to $\boldsymbol{C}$ and $\boldsymbol{Q}$ respectively.

3.2.3 Coattention Encoder

The coattention encoder learns co-dependant representations of the question and the document. We have implemented the encoder as described in the DCN paper [3]. We first compute $\boldsymbol{L} = \boldsymbol{C}^\top \boldsymbol{Q} \in \mathbb{R}^{(m+1)\times(n+1)}$. $\boldsymbol{L}$ will be used to calculate attention distribution in both direction, across the document for each word in the question ($\boldsymbol{A}^Q$) and across the question for each word in the context ($\boldsymbol{A}^C$). The question-to-context attention output ($\boldsymbol{S}^Q$) and coattention output ($\boldsymbol{S}^C$) are computed as following:

$$\boldsymbol{A}^Q = \text{softmax}(\boldsymbol{L}) \in \mathbb{R}^{(m+1)\times(n+1)} \quad \text{and} \quad \boldsymbol{A}^C = \text{softmax}(\boldsymbol{L}^\top) \in \mathbb{R}^{(n+1)\times(m+1)}$$
$$\boldsymbol{S}^Q = \boldsymbol{C}\boldsymbol{A}^Q \in \mathbb{R}^{(l)\times(n+1)} \quad \text{and} \quad \boldsymbol{S}^C = [\boldsymbol{Q}; \boldsymbol{S}^Q]\boldsymbol{A}^C \in \mathbb{R}^{2l\times(m+1)}$$
$$\boldsymbol{u}_t = \text{Bi-LSTM}(\boldsymbol{u}_{t-1}, \boldsymbol{u}_{t+1}, [\boldsymbol{c}_t; \boldsymbol{S}_t^C])$$

We then pass the resulting hidden states $\boldsymbol{U} = [\boldsymbol{u}_1 \cdots \boldsymbol{u}_m] \in \mathbb{R}^{2l\times m}$ known as coattention encoding to the decoder module to predict answer span.

3.2.4 Dynamic Pointing Decoder

To allow the model to recover from initial local maxima corresponding to an wrong answer, we have implemented an iterative multi-pass technique to predict the start and end point of the answer span as described in the original DCN paper [3]. Dynamic decoder is implemented as LSTM based state machine. At iteration i, the previous hidden state of the LSTM, $\boldsymbol{h}_{i-1}$ along with the coaatenntion encoding corresponding to the previous estimate of the start and end position $[\boldsymbol{u}_{s_{i-1}}; \boldsymbol{u}_{e_{i-1}}]$ are fed into the LSTM.

$$\boldsymbol{h}_i = \text{LSTM}_{dec}(\boldsymbol{h}_{i-1}, [\boldsymbol{u}_{s_{i-1}}; \boldsymbol{u}_{e_{i-1}}])$$

Given $\boldsymbol{h}_i$, $\boldsymbol{u}_{s_{i-1}}$, $\boldsymbol{u}_{e_{i-1}}$, we then feed them into Highway Maxout Network (HMN) to calculate α_t and β_t, the start and end score assigned to the tth word in the context. Then s_i and e_i denoting the estimate of the start and end position at iteration i, are set to positions having the maximum start and end score respectively.

$$\alpha_t = \text{HMN}_{start}(\boldsymbol{u}_t, \boldsymbol{h}_i, \boldsymbol{u}_{s_{i-1}}, \boldsymbol{u}_{e_{i-1}}) \qquad \beta_t = \text{HMN}_{end}(\boldsymbol{u}_t, \boldsymbol{h}_i, \boldsymbol{u}_{s_{i-1}}, \boldsymbol{u}_{e_{i-1}})$$
$$s_i = \arg\max_t (\alpha_1, \cdots, \alpha_m) \qquad e_i = \arg\max_t (\beta_1, \cdots, \beta_m)$$

The maxout layer act as an ensemble technique that combine the predictions of multiple models to form a final prediction. The HMN model is described as:

$$\text{HMN}(\boldsymbol{u}_t, \boldsymbol{h}_i, \boldsymbol{u}_{s_{i-1}}, \boldsymbol{u}_{e_{i-1}}) = \max\left(\boldsymbol{W}^{(3)}\left[\boldsymbol{m}_t^{(1)}; \boldsymbol{m}_t^{(2)}\right] + \boldsymbol{b}^{(3)}\right)$$
$$\boldsymbol{r} = \tanh\left(\boldsymbol{W}^{(D)}\left[\boldsymbol{h}_i; \boldsymbol{u}_{s_{i-1}}; \boldsymbol{u}_{e_{i-1}}\right]\right) \quad \boldsymbol{m}_t^{(1)} = \max\left(\boldsymbol{W}^{(1)}\left[\boldsymbol{u}_t; \boldsymbol{r}\right] + \boldsymbol{b}^{(1)}\right) \quad \boldsymbol{m}_t^{(2)} = \max\left(\boldsymbol{W}^{(2)}\boldsymbol{m}_t^{(1)} + \boldsymbol{b}^{(2)}\right)$$

As suggested in the DCN paper we have set the maximum number of iteration to 4 and used a max pool size of 16 in running our experiments [3].

3.2.5 Loss Function

We have used cumulative cross entropy of the start and end points across all iteration as the loss function: $\mathcal{L}_s = \frac{1}{N}\sum_{i=1}^{N} \text{CE}(y_{s_i}, \hat{y}_{s_i})$, $\mathcal{L}_e = \frac{1}{N}\sum_{i=1}^{N} \text{CE}(y_{e_i}, \hat{y}_{e_i})$, $\mathcal{L}_{total} = \mathcal{L}_s + \mathcal{L}_e$. The iterative procedure to estimate the start and end position stops if the maximum number of iteration allowed is exhausted or the prediction no longer changes.

3.3 QANet Overview

We have implemented the QANet model, which is is a feedforward model consisting of convolution and self attention [2]. The QANet model consists of five layers, including an embedding layer, an embedding encoder layer, an attention layer, a model encoder layer and an output layer. The main module of QANet model is an Encoder Block, a stack of following building blocks: positional encoding, residual connection, layer normalization, a stack of convolution layers, multi-head attention layer [15] and feed forward layer, as illustrated in Figure 1.

3.3.1 Embedding Layer

We have used the same embedding layer as described for the DCN model.

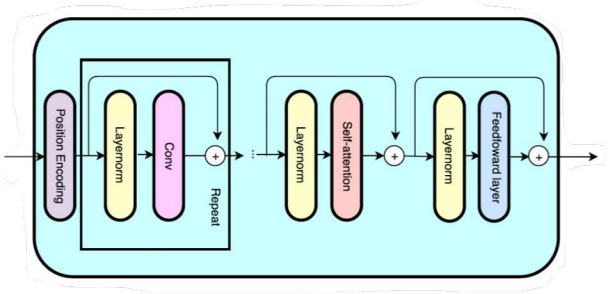

Figure 1: One Encoder Block [2]

3.3.2 Embedding Encoder Layer

We feed the output of embedding layer $\boldsymbol{W}^C$ and $\boldsymbol{W}^Q$ into a single encoder block. The encoder block for context words and question words share the same weights. We used the configuration suggested by the original QANet model for our encoder block. The kernel size is 7, the number of filters is 128, the number of attention heads is 8 and the number of convolution layers within a block is 4.

3.3.3 Attention Layer

For this module we used the same context query attention layer as the BiDAF model. [1].

3.3.4 Model Encoder Layer

Model encoder is a stack of 7 encoder blocks. 3 model encoders are chained together where the output of the first model encoder $\boldsymbol{M}_1$ is fed into the second one. Similarly the output of the second model encoder $\boldsymbol{M}_2$ is feed into the third model encoder which produces $\boldsymbol{M}_3$. The weights are shared between each of the 3 model encoder. For the encoder block, the parameter configuration is the same Embedding Encoder layer except that the number of convolution layer is 2 with in a block. $\boldsymbol{M_1}$, $\boldsymbol{M_2}$, $\boldsymbol{M_3}$ are then fed into the output layer.

3.3.5 Output Layer

This layer predicts the probability of each position in the context being the start or end of an answer span. The probabilities of the starting $\boldsymbol{p}^1$ and ending position $\boldsymbol{p}^2$ are modeled as following:

$$\boldsymbol{p}^1 = \text{softmax}\left(\boldsymbol{W_1}[\boldsymbol{M_1}; \boldsymbol{M_2}]\right) \quad \text{and} \quad \boldsymbol{p}^2 = \text{softmax}\left(\boldsymbol{W_2}[\boldsymbol{M_1}; \boldsymbol{M_2}]\right)$$

where $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are two trainable weight matrices.

3.3.6 Loss Function

We used sum of cross entropy loss of the predicted start and end distribution.

$$\mathcal{L}_s = -\frac{1}{N}\sum_{i=1}^{N}\left[\log(\boldsymbol{p}^1_{y_i^1}) + \log(\boldsymbol{p}^2_{y_i^2})\right]$$

where y_i^1 and y_i^2 are the ground truth position of start and end index of answer span of i^{th} example.

3.4 Term frequency and part-of-speech tags

Furthermore for additional input features in the embedding layer, as described in Chen et al 2017 [16], we added normalized term frequency and part-of-speech tags for each word. The normalized term frequency represents the number of times a word appears in the context or question (represented as a normalized float), and the part-of-speech tag is represented as a one-hot encoding over the 45 different part-of-speech tags. The part-of-speech tag is gathered from the nltk Python library, using the UPenn tagset.

4 Experiments

4.1 Data

The official Stanford Question Answering dataset SQuAD 2.0 will be used as the dataset [4]. It is a widely used large hand annotated dataset, which has the spans in the reference document as the answers. The questions and references have a decent level of diversity and sophistication. Finding the answer requires various forms of reasoning which may involve multiple sentences. Below is a sample example for SQuAD dataset.

The SQuAD training data provided in `data/train*` were used to train the models, and the dev sets `data/dev*` provide evaluation metrics for the models. The `train_eval` set contains 300.7 MB of training data, and the `dev_eval` set contains 15.8 MB of dev set evaluation data.

Context	The further decline of Byzantine state-of-affairs paved the road to a third attack in 1185, when a large Norman army invaded Dyrrachium, owing to the betrayal of high Byzantine officials. Some time later, Dyrrachium—one of the most important naval bases of the Adriatic—fell again to Byzantine hands.
Question	When did the Normans attack Dyrrachium?
Ground Truth Answers	1185, in 1185, 1185

Table 1: (Context, Question, Answer) example from SQuAD dataset [4]

4.2 Evaluation method

As the evaluation metric for the default project, we use the F1 score and Exact Match (EM) to evaluate the performance of our models on the SQuAD 2.0 dataset.

4.3 Experimental details

We implemented the DCN and QANet models. In running our experiments, we tried to respect model configurations suggested in the original papers [3], [2]. For both models we used pre-trained GloVe word vectors for word embedding, and implemented character level embedding to better handle out-of-vocabulary words. The dimension of the character embedding vector is set to 64.

For the DCN model, we have randomly initialized all LSTMs weights and set initial state to zero. For the dynamic decoder layer, we set number integration to 4 and maxout pool size to 16 as suggested in the original paper[3]. We apply dropout to regularize the network and ADAM optimizer with $\beta_1 = 0.8$, $\beta_2 = 0.999$ and constant learning rate $\alpha = 0.5$ to preform SGD to minimize the loss function during training. We used batch size of 64, hidden size of 100 with dropout rate of 0.2 and trained the model for 30 epochs which took around 36 hours. We found that the DCN model takes a long time to train, likely due to the presence of RNNs in the model which sequentially process the data, and due to the larger number of parameters in the decoder step's maxout layer. Additionally, the iterative process for estimating the answer span makes this layer computationally unparallelizable.

For the QANet model, we set the hidden size and the convolution filter size to 128. The embedding and modeling encoders have 4 and 2 convolution layers with kernel size 7 and 5, respectively. We use ADAM optimizer with $\beta_1 = 0.8$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$. As suggested in the original paper, we use a learning rate warm-up scheme with an inverse exponential increase from 0.0 to 0.001 in the first 1000 steps. We used batch size of 16, hidden size of 128 with dropout rate of 0.01 and trained the model for 30 epochs taking around 13 hours. We attempted to increase our model capacity by increasing the dimension of embedding vectors, hidden size, or the number of layers in the encoder block. However, due to limited hardware resources and lack of sufficient GPU memory, our attempts of increasing model size failed to generate increased performance.

We implemented our models in Python using Pytorch and carry our our experiments on Azure VMs featuring NVIDIA's Tesla V100 GPU.

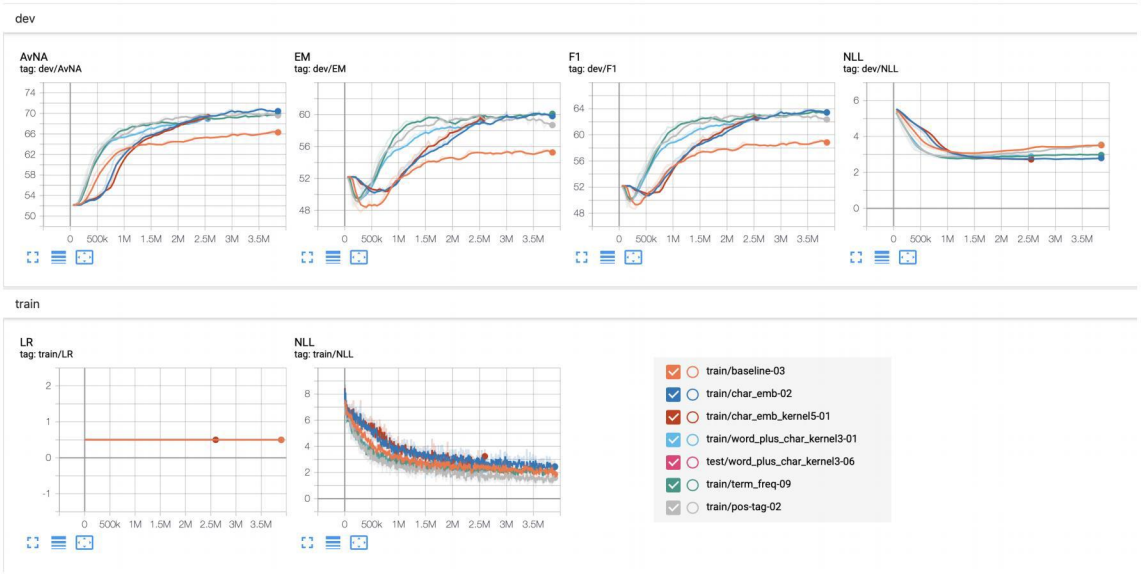

Figure 2: Experimental results of using character embeddings with kernel size 3 (char_emb-02), character embeddings with kernel size 5 (char_emb_kernel5-01), word + character embeddings (word_plus_char_kernel3-01), normalized term frequency (term_freq-09), and part-of-speech tags (pos_tag-02).

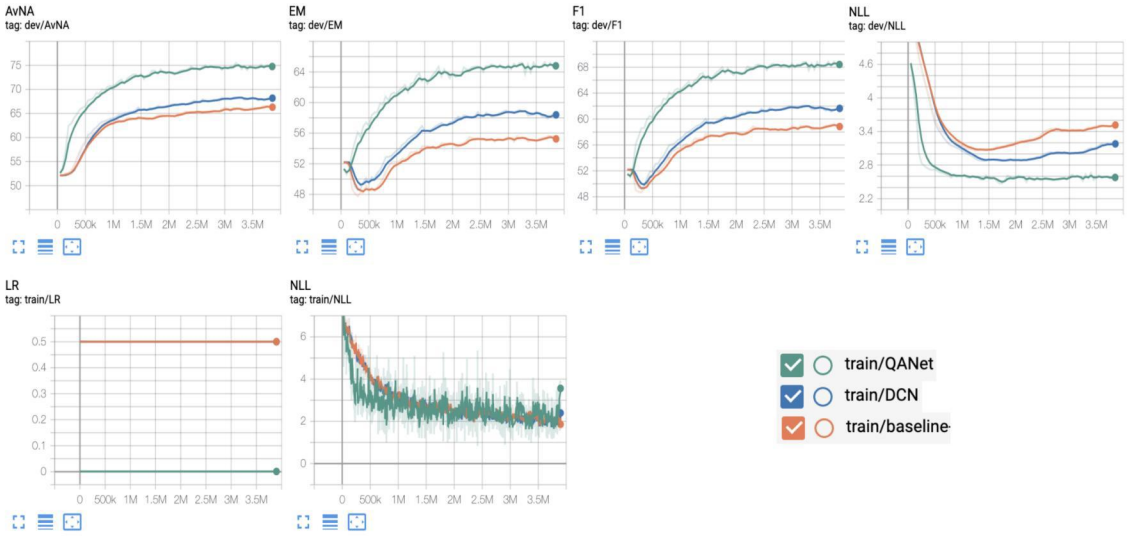

Figure 3: Experimental results of using the dynamic coattention network (dcn) and QANet transformer model (qanet) with character embeddings.

4.4 Results and Analysis

We analyzed the performance of our models and its ablations on the SQuAD 2.0 development set as illustrated in Table 2. Character embeddings improved the EM and F1 scores, whereas the additional input features sped up performance gains during training while only modestly improving overall performance.

The dynamic coattention network performed roughly the same as the baseline with character embeddings, while the QANet transformer model performed at a much higher level of accuracy than the other models.

One of the outcomes we noticed was that final performance (F1/EM scores) of the models sometimes differed little between experimental runs. For example, word-plus-character level embeddings performed roughly as well as character-only level embeddings (63.31 vs. 62.754 in F1 score, respectively after about 20 epochs). However, because the

Model	EM	F1
Baseline	55.60	59.24
Character embeddings	59.67	63.31
Word + character embeddings	59.486	62.754
Term frequency	60.23	63.54
Part-of-speech tags	59.28	62.62
Dynamic coattention decoder	55.73	59.72
Dynamic coattention decoder + char embeddings	58.28	61.82
QANet + char embeddings	65.081	68.76

Table 2: Experimental results on the development set

Model	EM	F1
Leaderboard submission	62.671	66.005

Table 3: Final Test Leaderboard results with QANet model with character embeddings (submitted as "Team 78")

combination of word and character embeddings resulted in faster performance gains in earlier epochs during training (faster than character-only and word-only embeddings in the baseline), it seems that the use of both character and word-level embeddings adds real value and information for helping the model to train. In practice, this helps us to be able to run experiments with a fewer number of epochs to speed up the iterative experimental process.

Simple additions to the model like character embeddings and feature engineering also appears to have improved model performance. For example, adding normalized term frequency information to the embedding [16] improved performance in a modest way (roughly 0.5 to 1 points of accuracy). Intuitively, term frequency helps because lesser-used words are probably more likely to be part of the correct answer than frequently-used words. Analytically, it seems that clever forms of feature engineering can help expose the model to helpful information more easily.

In analyzing the why the dynamic coattention network performed near the performance of the baseline BIDAF model, this may be due to the fact that both models already incorporate a 2-way attention between the context and the question. Furthermore, the stated purpose of the dynamic pointing decoder in the dynamic coattention network was to help the model recover from local maxima [3]. However, because the nature of local maxima is specific to the overall architecture and hyperparameters, the BIDAF approach of predicting start and end points for the answer may have been just as performant as alternating between the two dynamically.

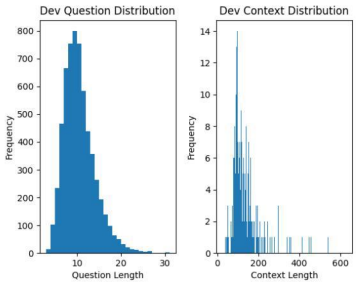

Figure 4: Question length (left) and context length (right) distribution of the development set

Performance Across Length: We analyzed how the DCN and QANet model's performance varies with respect to the length of context, question, and answer. As illustrated in Figure 5 and Figure 6, we do not observe significant performance degradation with longer context and question across models. This implies that the coattention module has the ability to focus on a small segment of the context and discard the non-relevant parts. Also on average, we can see that the QANet model is outperforming the DCN model with longer contexts and questions. Although our models are somewhat agnostic to long context and question, we do see notable performance deterioration with longer answers. The intuition behind this is that as the number words increases in the correct answer span, estimating correct answer span become more challenging.

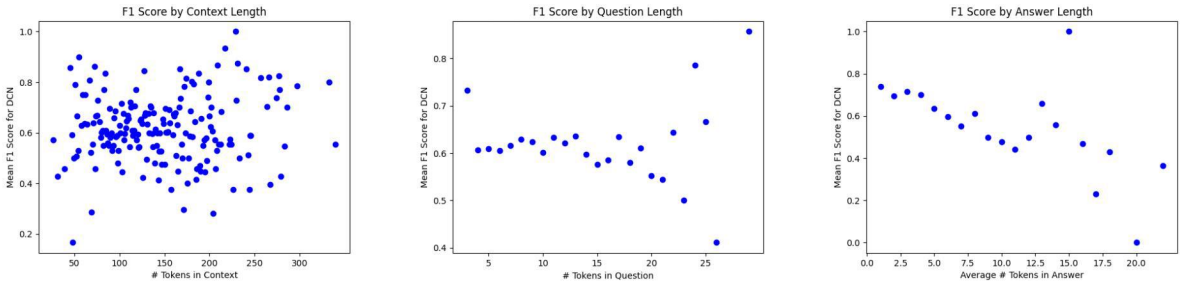

Figure 5: Performance of DCN for various lengths of contexts, questions and answers. The blue dot indicates the mean F1 at given length.

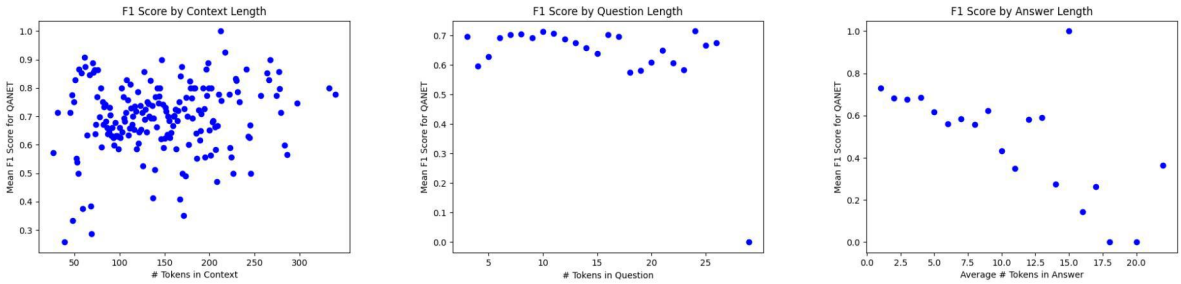

Figure 6: Performance of QANet for various lengths of contexts, questions and answers. The blue dot indicates the mean F1 at given length.

Performance Across Question Type: Non-factoid questions such as ('why' and 'how') inherently have a longer answer span than factoid questions such as ('when', 'where'). As our models' performance deteriorates with long answers , we should intuitively expect that they perform poorly with non-factoid questions. In fact, Figure 7 shows that mean F1 score for factoid questions ('when', 'who', 'where') exceeds non-factoid question such as ('why', 'how', 'what').

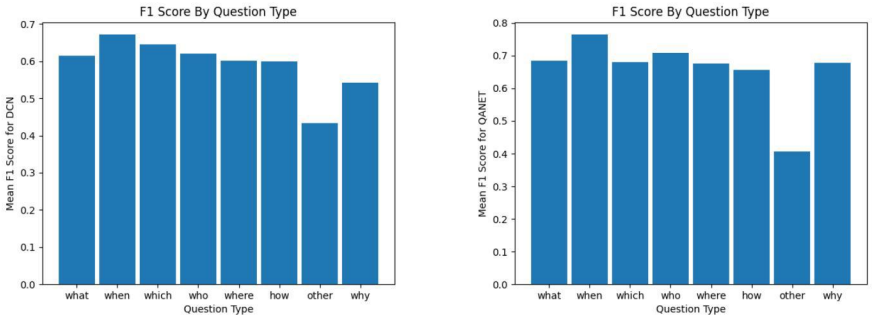

Figure 7: Performance of the DCN (left) and the QANet(right) across question types. The height of each bar represents F1 for the given questoin type.

5 Conclusion

In conclusion, we do not find significant performance gains for the dynamic coattention network on the question answering domain, although we do find that the QANet transformer based model achieves increased performance of an EM score of 62.671 and F1 score of 66.005 on the test leaderboard. Furthermore, additional input features like character embeddings and normalized term frequency helps the model achieve a significant 4.3 point F1 score increase.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *https://arxiv.org/abs/1611.01603*, 2016.

[2] M.-T. Luong R. Zhao K. Chen M. Norouzi A. W. Yu, D. Dohan and Q. V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In *https://arxiv.org/abs/1804.09541*, 2018.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. In *https://arxiv.org/abs/1611.01604*, 2016.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *https://arxiv.org/abs/1606.05250*, 2016.

[5] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[6] Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. Machine comprehension with syntax, frames, and semantics.

[7] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Association for Computational Linguistics (ACL)*, 2016.

[8] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Hierarchical question-image co-attention for visual question answering. In *https://arxiv.org/abs/1603.01547*, 2016.

[9] Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. In *https://arxiv.org/abs/1606.02245*, 2016.

[10] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *https://arxiv.org/abs/1606.00061*, 2016.

[11] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. In *https://arxiv.org/abs/1608.07905*, 2016.

[12] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems, pp. 2692–2700*, 2015.

[13] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Association for Computational Linguistics (ACL)*, 2014.

[14] Klaus Greff Rupesh Kumar Srivastava and J¨urgen Schmidhuber. Highway networks. In *https://arxiv.org/abs/1505.00387*, 2015.

[15] Niki Parmar-Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. Attention is all you need. 2017.

[16] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *arXiv preprint arXiv:1704.00051*, 2017.